CSE142 - General Debugging Guide

DISCLAIMER: This guide will not fix your bugs for you but will help you understand your errors. The causes listed in the tables below are "possible", but not guaranteed.

Common Compiler Errors:

Compiler errors are errors that prevent a program from compiling and are usually a result of minor syntax issues that are *generally* easy fixes as long as you understand the error message and can find where in the program it is occurring. Luckily, line numbers that are in the vicinity of where the bug is (most of the time) are attached to the error. Below is a table of common error messages that you might encounter in CSE 142.

Error Message	"Possible" Cause
Cannot find symbol — class	Your file is not saved in the same folder as the file that refers to it.
Cannot find symbol — method	The method name is wrong and/or it is nonexistent
Cannot find symbol — variable	You are using an undeclared variable or variable name is misspelled
Class, interface, or enum expected	Too many closing braces }
Illegal start of expression	Missing closing braces } for a method or extra parentheses ()
Reached end of file while parsing	Missing closing braces } at end of file
<*???*> expected	Missing <*???*> at area of line number
Class is public, should be declared in a file…	Class name does't match the saved filed name (case sensitive!)
Incompatible types — expected *type*	Your types are not matching up with what's given and what's expected
Missing method body	Method declaration line has a semicolon ;
Missing `return` statement	There's a path with no `return` value
Unreachable statement	There's code written after a `return` statement.
Variable might not have been initialized	A variable you are using might not have an assigned value.
Unexpected type — required variable	You used = instead of ==
Unclosed `string` literal	Missing "" around a `string`

Common Runtime Errors:

Runtime errors are errors that occur when the program is running. Your program may compile successfully but still have errors, so watch out for these!

Error Message	"Possible" Cause
No error message! My program appears to be frozen!	You have an infinite loop somewhere
`StringIndexOutOfBoundsException`	You're accessing a character in a `String` with an index that is nonexistent
`ArrayIndexOutOfBoundsException`	You're accessing an array element with an index that is nonexistent
`NullPointerException`	You're accessing a `null` value (check every period . or open bracket [on line of error)
`InputMismatchException`	Your `Scanner` is reading the wrong type
`NoSuchElementException`	You're reading past the end of a `Scanner`

Caveats With Error Messages:

- Compiler does not always report the real error
- Error message may seem nonsensical
- Line numbers given in error messages may not be correct
- There may be multiple errors

Debugging Strategies:

In cases where you encounter a bug that cannot be fixed using the table above or a situation where there is no error message (logic error), you can't just blindly stare at the code and hope the bug will pop out to you. It's also not a good strategy to just "shotgun" every line of code and hope the random changes you make will fix the problem. There are a variety of ways to debug a program but based on the type of bug you run into, you may want to consider different approaches. Below are some of the most effective ways of debugging.

JGrasp Debugger

JGrasp has a really nice debugger that is extremely helpful for the majority of bugs you will encounter. Here is a useful guide on how to use the debugger. The debugger allows you to step through your code line by line until you hit a breakpoint. The reason why this is useful is because the debugger lays out values held in all variables and presents what's happening visually. It can better your understanding of what the program is doing, and often point the way to the mistake.

`println` Statements

Another way to examine the state of the program is by printing out various states of the program. By "logging" important values and information at various parts of the program, you know when, where, and how the program is affecting the output. A common way of using `println` statements is by marking the area surrounding the potential problem so you know what's happening before and after. Make sure you remember to remove these debugging `println` statements after you find your bug!

Start Small

An effective way to localize errors is to develop the program in bits. Start small and add little pieces of code incrementally, testing every time in between. If there is an error, it's very likely that it originated from the last piece of code inserted. This will make bug hunting a lot easier and faster. Another upside is that adding small amounts of code usually leads to fewer errors at a time so you won't get a long list of errors.

Commenting

Commenting pieces of the code out is a good way to figure out which parts of your program is causing the error. If you suspect somewhere in the code is creating the error, comment it out and see if you are still getting the error. If you aren't, voila! If you are, then you know that the bug is likely in another part of the program and you can rinse and repeat until you find where the error is occurring.

Take A Break

If you've been wrestling with a bug for a while, stop and take a break! Let your brain rest. The more frustrated you are, the harder it is to find bugs. After taking a break, it can be easier to spot syntax errors, as well as logic errors.